ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 4

PROJECT DATE : 14.07.2020

GROUP NO : G22

GROUP MEMBERS:

150180903 : Khayal HUSEYNOV

150180901 : Ramal SEYIDLI

150180725 : İsmayil Buğra KÜTÜKOĞLU

150180720 : Kerim GENÇ

Contents

1 INTRODUCTION

In this project, we designed a software-based (microprogrammed) control unit for a basic computer which has a very similar architecture that we worked in the previous projects. We updated our project 2 and fixed the mistakes we made in project 3.

2 PROJECT PARTS

2.1 MAP

Our mapping logic is similar to the one given in the lecture slides. We add two zeros to the right side of our address. We add one bit to the left side of our address as well, but it is one instead of zero. We added one so that our program can start with a FETCH (0 000000 00) operation. which can be seen from the below figure.

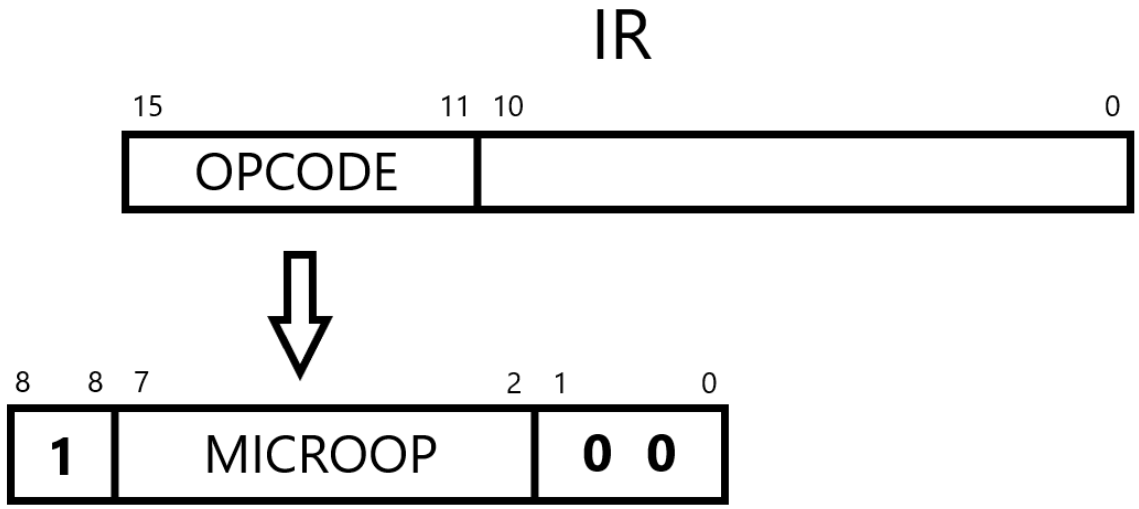


Figure 1: Conversion Logic

Following is the logisim implementation of the logic described.

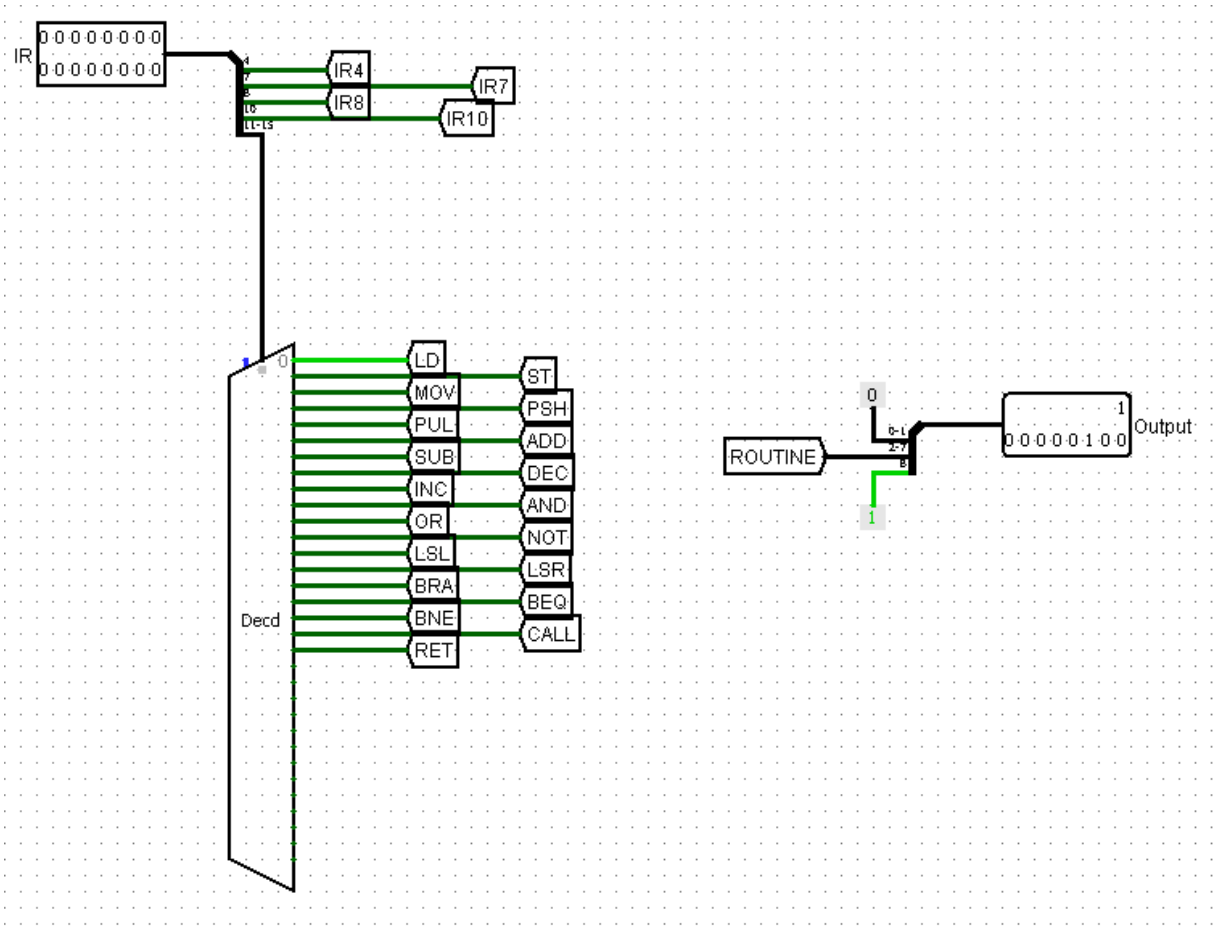


Figure 2: Logic of the Mapping Proccess

Then we mapped opcodes to microoperations according to select bits such as IR4, IR7, IR8, IR10.

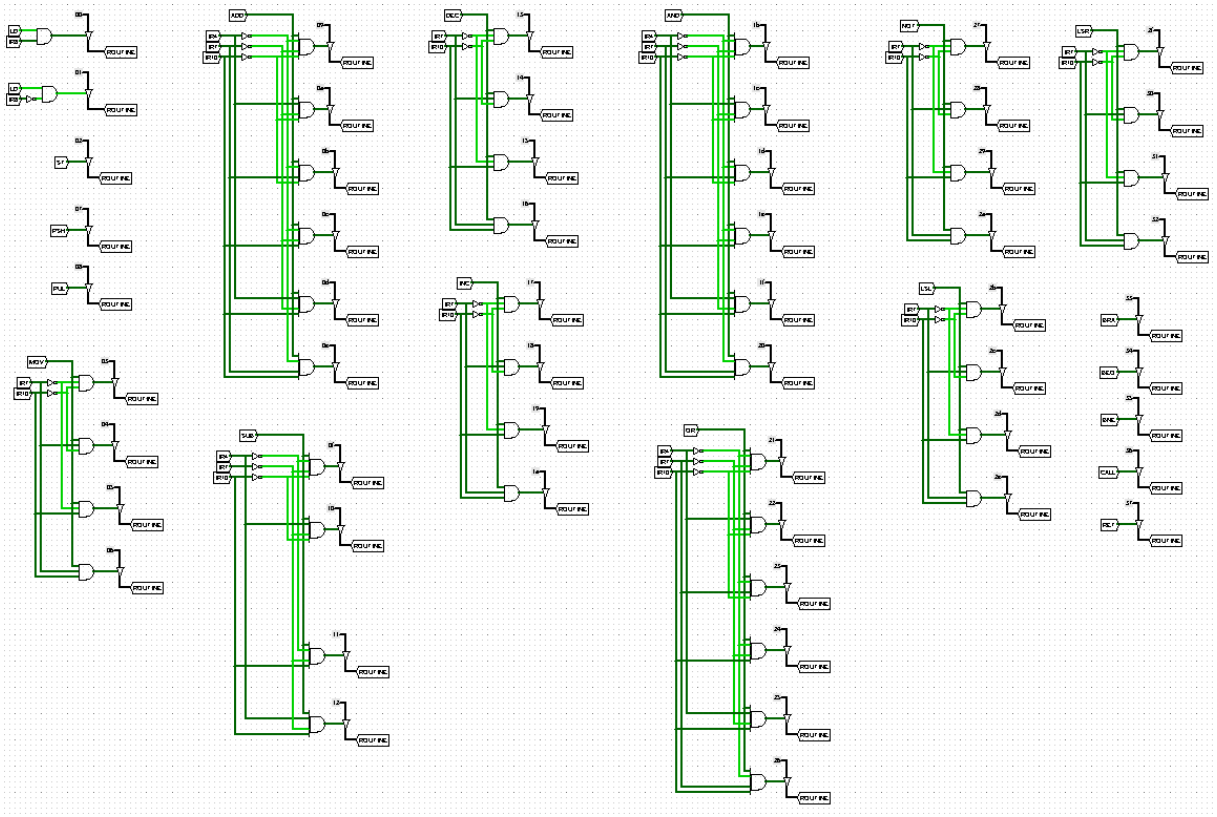
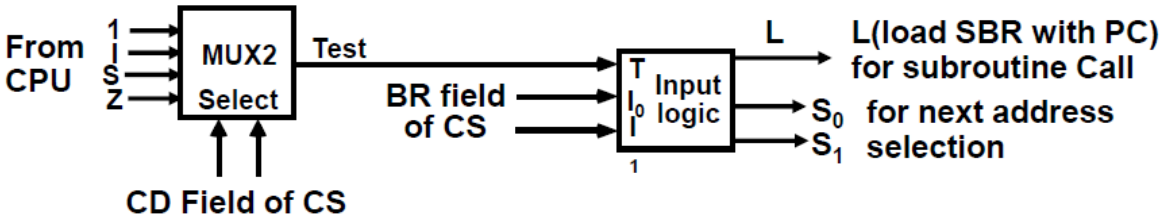


Figure 3: Mappings

2.2 BRANCH LOGIC

Our branching logic is similar to the one described in the lecture slides. The only difference are the conditions. We still use 2 bits for conditions but we only have 3 of them in our design. They are Unconditional, Zero, and NotZero, respectively.



Input Logic

I_0I_1T	Meaning	Source of Address	S_1S_0	L
000	In-Line	CAR+1	00	0
001	JMP	CS(AD)	10	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	10	1
10x	RET	SBR	01	0
11x	MAP	DR(11-14)	11	0

$$S_0 = I_0$$
$$S_1 = I_0I_1 + I_0'T$$
$$L = I_0'I_1T$$

Figure 4: Logic of Branching taken from Slides

As you can see from Logisim implementation below, our condition test's 11 input of multiplexer is left empty.

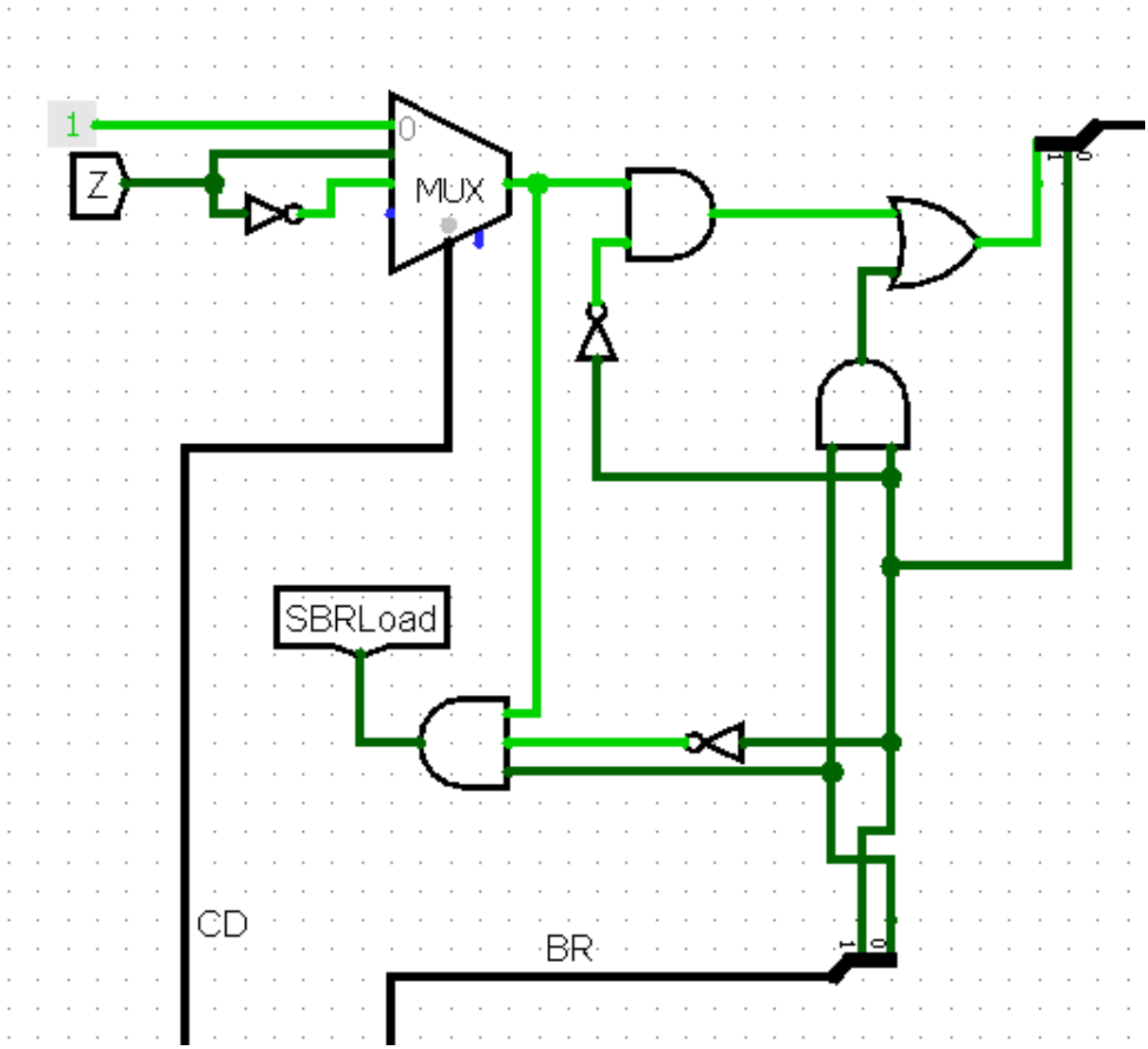


Figure 5: Logisim Implementation of Branch Logic

2.3 ROMS

We knew we had to use a ROM to store our microprograms, but we thought this was still not efficient enough. We still had to link the microprograms to control signals by hand. We had discovered from the previous project that this was too cumbersome of a task to do. We were worried of the errors that would come about from connecting a wrong wire to an unwanted place.

We decided on using another ROM to store our control signals as well. We felt like this was the exact purpose of a microprogrammed control unit.

2.3.1 MICROPROGRAMS

We decided to use an excel sheet for the instructions of microprograms. The sheet is named microprograms.xlsx and is provided as part of the report.

000	76001	78002	7a003	78600	70400	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
010	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
020	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
030	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
040	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
050	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
060	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
070	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
080	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
090	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
0a0	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
0b0	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
0c0	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
0d0	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
0e0	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
0f0	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
100	02000	00000	00000	00000	04000	00000	00000	00000	06000	00000	00000	00000	08000	00000	00000	00000
110	0a000	00000	00000	00000	0e000	00000	00000	00000	10000	00000	00000	00000	1211d	14000	00000	00000
120	16121	18000	00000	00000	1a000	00000	00000	00000	1c000	00000	00000	00000	1e000	00000	00000	00000
130	20000	00000	00000	00000	22000	00000	00000	00000	24000	00000	00000	00000	26000	00000	00000	00000
140	28000	00000	00000	00000	2a000	00000	00000	00000	2c000	00000	00000	00000	0814d	2e14e	30000	00000
150	0a151	2e152	30000	00000	0e155	32156	34000	00000	10159	3215a	34000	00000	0815d	3615e	30000	00000
160	0a161	36162	30000	00000	0e165	38166	34000	00000	10169	3816a	34000	00000	3a000	00000	00000	00000
170	3c000	00000	00000	00000	3e000	00000	00000	00000	40000	00000	00000	00000	42000	00000	00000	00000
180	44000	00000	00000	00000	46000	00000	00000	00000	4e000	00000	00000	00000	50000	00000	00000	00000
190	52000	00000	00000	00000	54000	00000	00000	00000	56000	00000	00000	00000	58000	00000	00000	00000
1a0	5a000	00000	00000	00000	5c000	00000	00000	00000	5e000	00000	00000	00000	60000	00000	00000	00000
1b0	62000	00000	00000	00000	64000	00000	00000	00000	66000	00000	00000	00000	68000	00000	00000	00000
1c0	6a000	00000	00000	00000	6c000	00000	00000	00000	6e000	00000	00000	00000	00204	00000	00000	00000
1d0	00a04	00000	00000	00000	01204	00000	00000	00000	721d9	14204	00000	00000	161dd	74000	00000	00000
1e0	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
1f0	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000

Figure 6: Microprograms Written into ROM1

2.3.2 MICROOPERATIONS

Our previous report was a mess so this time around we ported all of our control signal instructions to an another excel sheet. This sheet is named microoperations.xlsx and is also provided as part of the report.

00	000000	081050	080000	040060	083000	082000	000000	040d00	002d00	0400a0	000300	000200	081090	0d3000	093000	093000
10	050d00	010d00	010d00	0db000	09b000	058d00	018d00	180000	040000	000300	000000	100000	000200	0df000	09f000	09f000
20	05cd00	01cd00	01cd00	0e3000	000000	000000	000000	0a3000	0a3000	060d00	020d00	020d00	0cb000	08b000	048100	008d00
30	0eb000	0ab000	068d00	028d00	0ef000	0af000	06cd00	02cd00	000500	0000a0	000990	00001f	000200	000017	000000	000000

Figure 7: Control Signals Written into ROM2

After porting all of the control signal instructions we realized that 5 of them rely on

Instruction Register to do the respective operation. So we took them out (OutASel, OutBSel, RegSelRF, RegSelARF, OutCSel) and implemented them by hand.

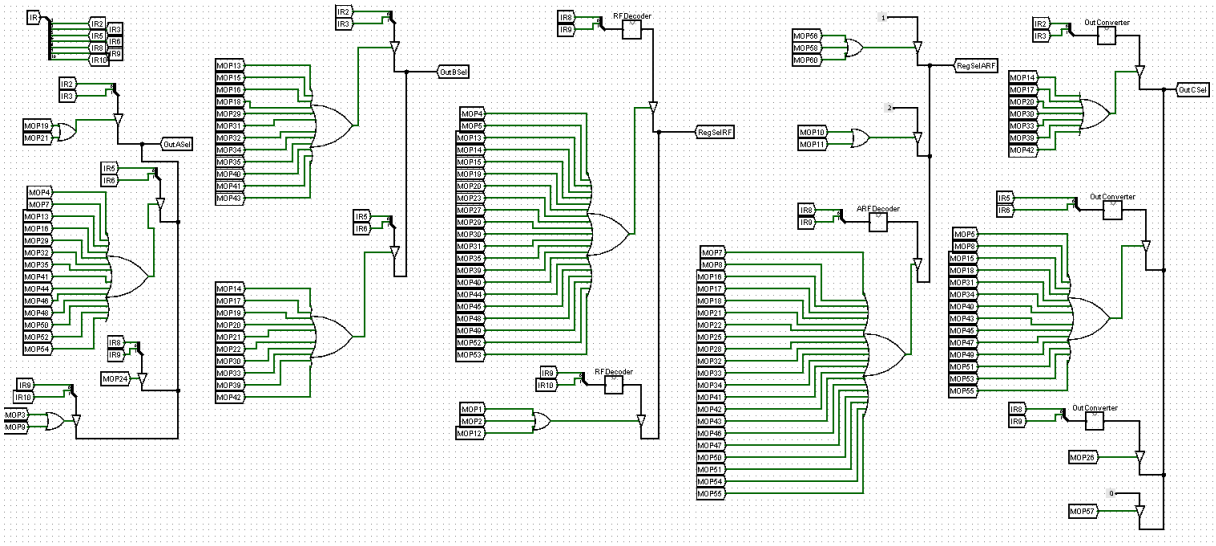


Figure 8: Hardwired Control Signals

3 RESULTS

3.1 TEST1

EXAMPLE

The code given below adds data that are stored at M[A0]+M[A1]+M[A2]+M[A3]+M[A4] and stores the total at M[A5]. It is written as a loop that iterates 5 times.

You have to determine the binary code, write it into memory, and execute all these instructions.

```
        ORG  0x20          # Write the program starting from the address 0x20
        LD R0 IM 0x05      # R0 is used for iteration number
        LD R1 IM 0x00      # R1 is used to store total
        LD R2 IM 0xA0
        MOV AR R2          # AR is used to track data adrress: starts from 0xA0
LABEL:  LD R2 D            # R2 <- M[AR] (AR = 0xA0 to 0xA4)
        INC AR AR          # AR <- AR + 1 (Next Data)
        ADD R1 R1 R2       # R1 <- R1 + R2 (Total = Total + M[AR])
        DEC R0 R0          # R0 <- R0 – 1 (Decrement Iteration Counter)
        BNE IM LABEL       # Go back to LABEL if Z=0 (Itertaion Counter > 0)
        ST  R1 D           # M[AR] <- R1 (Store Total at 0xA5)
```

Figure 9: Symbolic Representation of the Test Case

This test case was given to us in the instruction pdf and we converted all of the values to their binary representation and then converted those binary representations into hex digits so we can write them into our RAM for testing.

Figure 10: Converted Values Written into RAM

We observed that our basic computer added the data stored in M[A0], M[A1], M[A2], M[A3], and M[A4] together and stored the outcome to M[A6], not M[A5] as the example description suggested. We then did the calculations by hand and observed that, in fact, our basic computer did the additions correctly.
As can be seen from the figure, we assigned some numbers from M[A0] to M[A4] and run the test. The value of M[A6], as expected, was 0xFA.

3.2 TEST2

The test case provided by the pdf was not enough to test all of the operations that our basic computer can perform. So we wrote some of our own to check for mistakes.

```
LD R0 IM 0x00 (0x0000)
LD R1 IM 0x01 (0x0201)
LD R2 IM 0x00 (0x0400)
LD R3 IM 0x03 (0x0603)
SUB R2 R3 R1   # R3-R1 = R2 = 0x02 (0x322C)
AND R0 R2 R1  # R2 AND R1 = 0x00 (0x4844)
OR R3 R1 R3 # R1 OR R3 = R3 = 0x03 (0x532C)
NOT R0 R3 # R0 <- NOT R3 = 11111100 = 0xFC (0x5860)
LSL R2 R0 # R2 <- LSL R0 = 11111000 = 0xF8 (0x6200)
LSR R2 R2 # R2 <- LSR R2 = 01111100 = 0x7C (0x6A40)
```

Figure 11: Symbolic Representation of the Test Case

```
00 00 00 02 01  04 00 06 03  32 2c 48 44  53 2c 58 60
10 62 00 6a 40  00 00 00 00  00 00 00 00  00 00 00 00
```

Figure 12: Converted Values Written into RAM

Our basic computer performed all of the instructions flawlessly and managed to get the results described through the comments of the symbolic representation figure.

3.3 TEST3

We still had some operations left over that were unchecked. So we wrote another simple test case just for those.

```
LD R0 IM 0xA0 # R0 = 0xA0 (0x00A0)
MOV SP R0 # SP = R0 = 0xA0 (0x1700)
PSH R0 # M[SP] = M[A0] = A0 (0x1800)
PUL R1 # R1 = A0 (0x2200)
CALL 0x20 # Goes to 0x20 (0x8820)
LD R3 IM 0x0B # R3 = 0B (0x060B)
RET # Returns (0x9000)
LD R2 IM 0x0C # R2 = 0x0C (0x040C)
```

Figure 13: Symbolic Representation of the Test Case

```
00 00 a0 17 00 18 00 22 00 88 20 04 0c 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 06 0b 90 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 14: Converted Values Written into RAM

Our basic computer performed all of the instructions flawlessly and managed to get the results described through the comments of the symbolic representation figure.

4 DISCUSSION

We revised our control signal instructions and fixed a lot of mistakes. We then divided these instructions to microoperations. We wrote microprograms that utilize these microoperations. We designed mapping logic that decides which microprogram should be performed depending on the instruction register. We wrote and stored our microprograms and microoperations in ROMs. We designed a logic called Branch Logic that chooses the source of the Control Address Register. First of all, we designed a way to implement all of the opcodes which are given in pdf file by deciding whenever the selected bits are '1' according to time signals. Secondly, we sorted this list according to input variables. Both of the lists are added to the report.

5 CONCLUSION

In this project, we designed the almost same basic computer with a microprogrammed control unit. This time we managed to implement all of the operations on time and had spare time to test our circuitry. To conclude, this project helped us learn how to work with ROMs and work efficiently.